Prof. Bernd Finkbeiner, Ph.D.
Dipl.-Inf. Rüdiger Ehlers
Dipl.-Inf. Hans-Jörg Peter
Michael Gerke, B.Sc.

Reactive Systems Group
Saarland University
Due: Thursday, 13th November 2008
before the lecture

Embedded Systems 08/09 – Problem Set 3

Problem 1 (Extending the stop-watch) (30 pts.)

In this exercise you should **incrementally extend** the stop-watch model from problem set 2. As a starting point, you can either use your own model (*as far as it works correctly*) or the following one:

`http://react.cs.uni-sb.de/~peter/stopwatch.mdl`

Extend the model in the following way:

1. Remove the reset-to-0 of counter values that is currently coupled to starting the stop-watch. Instead, implement the following reset procedure: the counters are reset to 0 iff a full store-and-recall cycle[1] is performed while the stop-watch is stopped. If the stop-watch is being stopped while the `store/recall` switch is engaged then disengaging the `store/recall` switch should just recall the stored value. It then needs a full further store-and-recall cycle while holding the stop-watch stopped for resetting it to 0. (15 pts.)

2. Up to now, your stop-watch reads the switches and updates its state just once every second, giving a notable delay on some operations (e.g., starting the stop-watch — count the number of transitions that happen until the first increment of the internal counters happens). Change your model such that it internally runs with 10 Hz clock frequency. (*Hint:* Don't forget to enlarge the time step of the simulator; otherwise, you will become very sleepy...) (15 pts.)

Your model must at least fulfill the following scenarios:

(i) Start → Store → Stop → Recall
(*You should see the stored time*)
Store → Recall
(*You should see time 0*)
Start
(*Time runs again from 0 on*)

(ii) Start → Store → (wait a few sec) → Lap → Stop
(*You should see the frozen time of the last Lap*)
Recall
(*You should see the stored time*)
Start

[1] by engaging and disengaging the `store/recall` switch

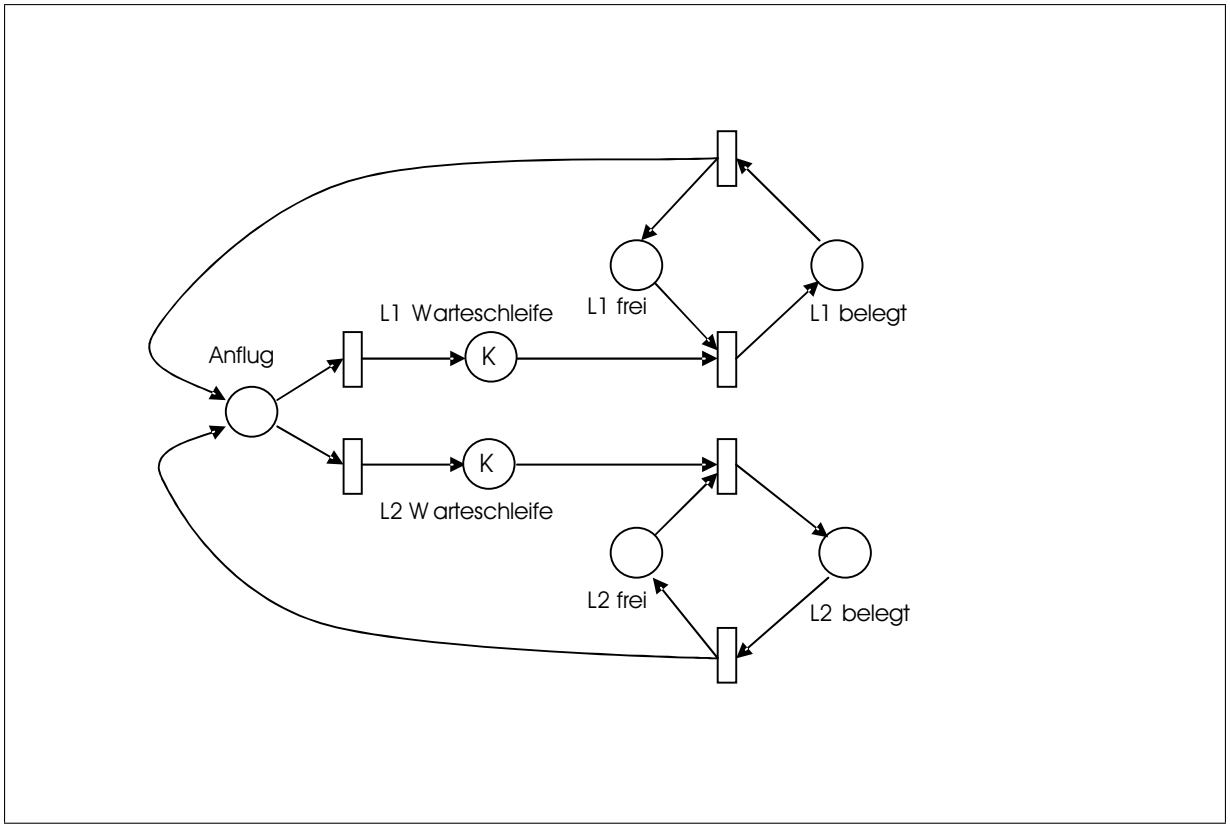

Figure 1: A Petri net modelling two landing strips.

(*The display remains frozen, the internal counter starts running again*)
Lap
(*The display shows the internal time and gets refreshed every second*)

(iii) Start → Lap
(*Display should freeze*)
(wait a few sec) → Store → Lap
(*You should see the internally elapsed time again*)
Recall
(*Time is reset to the stored time*)
Stop
(*Time should freeze*)
Store → Recall
(*You should see time 0*)

Please submit your MDL file to `es08@react.cs.uni-sb.de`

Problem 2 (Petri nets – part one) (30 pts)

Consider the net in Figure 1. It models the allocation of the landing strips of an airport. The airport has two strips exclusively reserved for landing airplanes. Each strip has a waiting list (Warteschleife) that can contain at most k airplanes. Furthermore, a strip can either be free (frei)

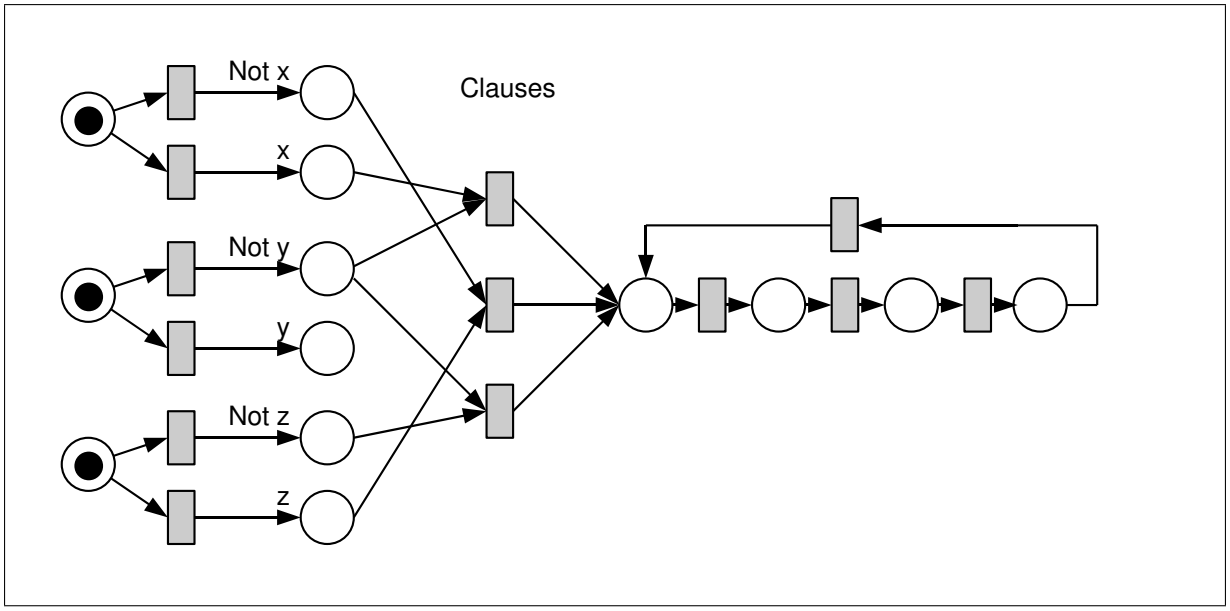

Figure 2: Example for problem 3

or occupied (belegt). When an airplane approaches (Anflug) it selects a strip. It is possible to use both strips at the same time. Initially, there are n airplanes flying.

1. Complete the net. Give the weights of the edges, the capacity of the places, and a meaningful initial marking. (15 pts.)

2. Modify the net so that the maximal number of airplanes in one waiting list (Warteschleife) is coded not in the capacity of the places, but rather in the initial marking. (15 pts.)

Problem 3 (Petri nets – part two) (40+30 pts)

Consider the following (closed) boolean formula $\forall x, y, z : (x \wedge \neg y) \vee (\neg x \wedge z) \vee (\neg y \wedge \neg z)$, which is in disjunctive normal form, preceded by universal quantifiers. Note that there is a simple transformation of this formula to a Petri net which can deadlock [2] if and only if this formula is **not** satisfiable [3]. The corresponding Petri net is shown in Figure 2.

Answer the following questions:

1. Explain why the Petri net can deadlock from the given initial marking if and only if the corresponding boolean formula is **not** satisfiable. (10 pts.)

2. Let such a universally quantified boolean formula in disjunctive normal form be given as a tuple $\mathcal{F} = \langle V, C \rangle$ with a set of variables V and a set of clauses $C \subseteq 2^{V \times \{-1,1\}}$. The set $\{-1, 1\}$ is used to mark each literal as a negated or non-negated occurrence of a variable.

   As an example, the formula $\forall x, y, z : (x \wedge \neg y) \vee (\neg x \wedge z) \vee (\neg y \wedge \neg z)$ can be represented as $\mathcal{F} = \langle \{x, y, z\}, \{\{(x, 1), (y, -1)\}, \{(x, -1), (z, 1)\}, \{(y, -1), (z, -1)\}\}\rangle$.

[2] i.e. there exists a sequence of transitions in this net such that at some point there are no transitions enabled

[3] note that the definitions of satisfiable and valid coincide here, since we are considering closed universally quantified DNF formulae

Your task is to describe a conversion that takes a universally quantified boolean formula $\mathcal{F}$ in disjunctive normal form and converts it to a place/transition Petri net $N = (P, T, F, K, W, M_0)$ such that N can deadlock if and only if $\mathcal{F}$ is not satisfiable. (20 pts.)

3. Take the simplified version of the example Petri net that is depicted in Figure 3. Build the *reachability graph* from the initial marking $\{1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 1, 4 \mapsto 0, 5 \mapsto 0, \ldots, 10 \mapsto 0\}$. Is the encoded formula satisfiable? (10 pts.)

4. (*Bonus question*) What hardness result for deadlock checking of Petri nets can be deduced from the above reduction? (10 pts.)

5. (*Bonus question*) Can you reduce the satisfiability problem of a universally quantified boolean formula in disjunctive normal form to the reachability problem of a certain marking? Justify your answer. (20 pts.)

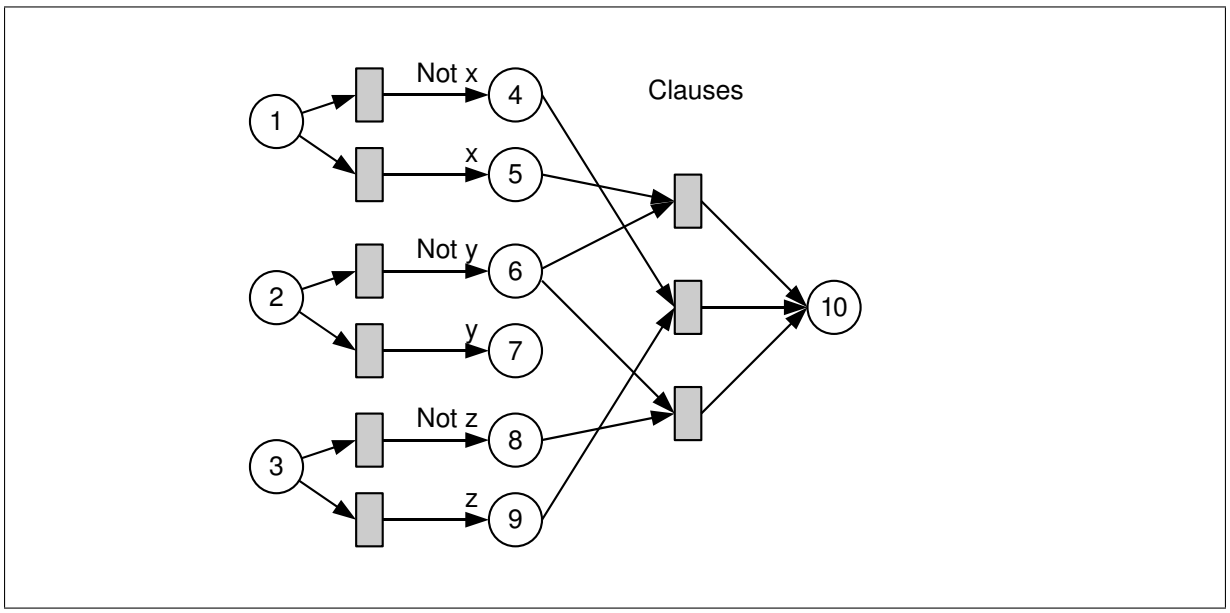

Figure 3: Simplified example for problem 3